



PROGRAMMING Expert

Mike James looks at the science behind Virtual Earth and shows you how, by manipulating it to customise maps for use in your own web pages, you can effectively put the whole world in your hands

PROGRAMMING NEWS

VBA SAFE FOR NOW

Microsoft spokesman Mark Quirk, group technical manager of the .NET Developer Group, confirmed that Visual Basic for Applications (VBA) will be an integral component of the next version of Office.

Given the demise of Visual Basic 6 and the move to .NET languages, the future of VBA had been in doubt. There were rumours it would not be included in the next version of Office. But Microsoft will allow it to coexist for now, though it won't be developed further and there will be no 64-bit version.

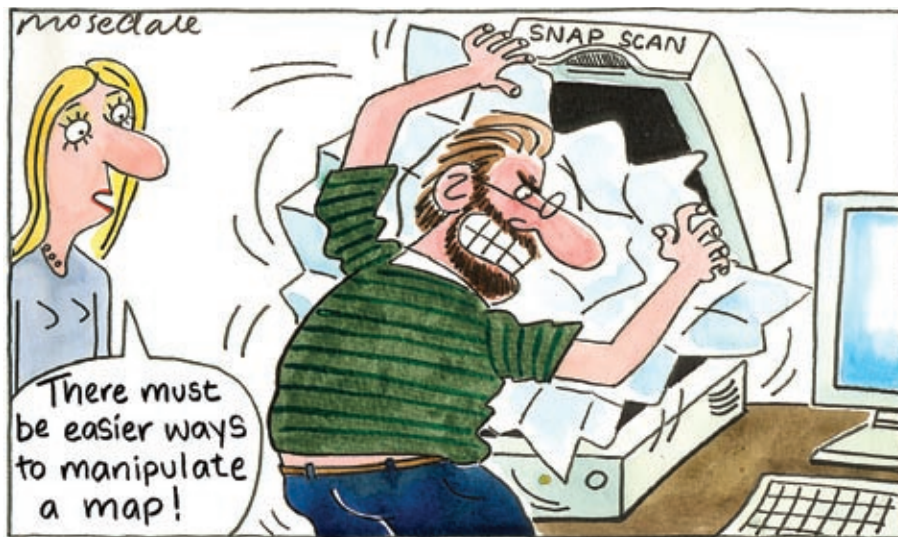
A new toolkit – VS Tools for the Microsoft Office System – is included with Visual Studio 2005, but we must wait for the next version of Office to see a fully integrated .NET-based macro facility. For details, see <http://msdn.microsoft.com/office/understanding/vsto>.

MICROSOFT LAUNCHES WEB APIs

To keep Google from cornering the web developer market, Microsoft has made new APIs available. The APIs allow access to MSN Web Search, Windows Desktop Search, MSN Virtual Earth and MSN Messenger. Web and desktop searches are similar to the equivalent Google products. Our project (right) shows you how to use the Virtual Earth API. Messenger API lets you add activities such as multiplayer games that can be run by Messenger users. Google's instant messenger is based on open-source standards, so it should be easier to work with. Find out more at <http://msdn.microsoft.com/msn>.

JAVA TEAM TACTICS

Borland's JBuilder development environment for Java has been updated. JBuilder 2006 allows programmers to collaborate on a project no matter where they are located. Local and remote programmers can work on code at the same time, allowing pair programming at a distance. It also comes with new extras such as Optimizait, to make your code go faster, and the CaliberRM requirements management tool. A free version without team programming features is available at www.borland.com.



People find maps fascinating and useful, so it's no surprise that Google, Yahoo! and MSN all provide online mapping tools and even satellite imagery. A new twist to these services is the addition of APIs that let programmers use the mapping systems within their own programs.

Unfortunately, Yahoo! and Google require you to register to gain access to their APIs and Google places additional restrictions on use that make it slightly more difficult to try out. MSN, on the other hand, allows casual access to its service, which is ideal if you just want to experiment with maps. It also uses a similar method to implement its API as Google. This involves the creation of a JScript object, and is a good example of this technique. By looking inside the script, we can find out how it works and use the same techniques in our own web pages.

This month's project explores the Virtual Earth service. Along the way it discusses using the Document Object Model (DOM), creating JScript objects, how the two interact, and how you can customise Virtual Earth.

WELCOME TO VIRTUAL EARTH

You can try out the mapping provided by Virtual Earth at <http://virtualearth.msn.com>. You'll find, though, that the mapping of Europe and the UK in particular isn't very good. At the time of writing Virtual Earth really means Virtual USA, but there is a suggestion that the coverage will be improved in the future.

To use the Virtual Earth data in your own applications, you start by visiting the Via Virtual Earth website, www.viavirtualearth.com. Here you will find all you need to get started. The whole API

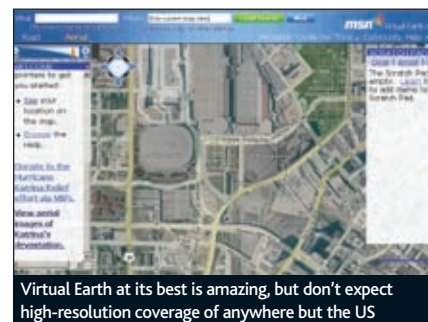
is based around a single JScript object that you can download and use, or access directly at <http://virtualearth.msn.com/js/mapcontrol.js>.

If you download MapControl.js and try to read it, you will discover that it is completely unformatted and very difficult to understand. If you want to examine the way it works, a tidier version with indenting is available at www.viavirtualearth.com/myvirtualearth/mapcontrol.js. You don't have to read the script, however, because it is fairly easy to use as long as you know its properties and methods.

GETTING STARTED

The easiest way to explain how it all works is to use the simplest mapping example possible. Use NotePad or any HTML editor to create the following HTML file:

```
<html>
<head>
<script src="http://
//virtualearth.msn.com/js/
```



Virtual Earth at its best is amazing, but don't expect high-resolution coverage of anywhere but the US



```
MapControl.js">#
</script>#
<script>#
function OnPageLoad()#
{#
    map = new VE_MapControl(53, 0,
        100, 'h',#
        "relative", 1, 1, 500, 300);#
    document.body.appendChild(map.
        element);#
}#
</script>#
</head>#
<body onLoad="OnPageLoad()">#
</body>#
</html>#
```

If you load the HTML page into a browser, you will see a view of 53N 0. The way that it works is fairly easy to understand. The first script tag loads the JavaScript code needed to create the MapControl. You can change this to point at a local version of the JavaScript if you download it. The OnPageLoad function first creates a new instance of the MapControl. The map.element property, which now comprises a DIV and its contents, is then added to the page using the DOM. A DIV is a container for HTML code, beginning with a <DIV> tag and ending with </DIV>. DIVs can be arranged on the page and formatted using Cascading Style Sheets (CSS). In this case, the DIV produced by MapControl contains a number of images that make up the map, and another DIV container that holds copyright information.

The MapControl constructor supplies all the details needed to create the map display. Its parameters are:

```
VE_MapControl(latitude, longitude,
    zoomLevel, mapStyle, position, x, y,
    width, height)#
```

You have to provide the latitude and longitude of the point in the middle of the map. If you want your program to locate street addresses, you will have to use a geocoding server to convert each address to a latitude and longitude. These services generally charge beyond a trial period; for more information search the web for a geocoding service.

The mapStyle parameter is r for road map, a for aerial photo and h for hybrid. The final parameters position and size the map on your web page. The position parameter is either absolute or relative, and if it is absolute the next two parameters give the position of the map. The final two give its size. You can see the effect of all these parameters by experimenting with the example given above.

METHODS AND PROPERTIES

Although the MapControl looks as if it is a JavaScript function, it is in fact an object. The VE_MapControl has a range of methods and properties that you can use to control it. For example, the SetCenter method will move the map location to the specified latitude and longitude.

Add a button to the web page:

```
<input id="Button1"#
    type="button"#
    value="button"#
    onclick="Button1_onclick()" />#
```

Also add the following click event handler:

```
function Button1_onclick()#
{#
    map.SetCenter(55, -1.5);#
}#
```

Clicking the button labelled Button re-centres the map on 55N, 1.5W. You can adjust all the other viewing parameters using similar methods. For more details, see the online documentation. There is even a PanToLatLon method that performs a simple animated flight to a new location.

As with other objects, there are a range of events that you can write code to respond to. To handle an event, all you have to do is define a function of the correct type and assign the function to the MapControls event property. For example, to handle the onMouseClick event you would first define the event handling function:

```
function OnClick(e)#
{#
    alert("Click")#
}#
```

Then you link the event handling function to the appropriate event property:

```
map.onMouseClick=OnClick;#
```

The above code means that each time the user clicks on the map, the message Click should pop up. To use the event information simply refer to the properties of the e parameter:

```
function OnClick(e)#
{#
    alert(e.latitude);#
}#
```

ADDING PUSHpins

As well as panning, zooming and generally controlling the map view it is also possible to add pushpin markers. The relevant method is:

```
AddPushpin(id, latitude, longitude,
    width, height, className, innerHtml)#
```

The pin has to be given a unique ID, a latitude and longitude to define its position, and a width and height to give its size. It is displayed as a DIV area on the page. The innerHTML parameter specifies any text you want to appear in this area. The area's size, background colour, border colour and so on are set by the CSS style given in className.

To define a blue pin with a red border that's 44x17 pixels you would use this stylesheet:

```
<style type="text/css" media=screen>#
.pin#
{#
    width:44px;height:17px;#
    font-family:Arial,sans-serif;#
    font-weight:bold;font-size:8pt;#
    color:White;overflow:hidden;#
    cursor:pointer;text-decoration:
        none;#
    text-align:center;background:
        #0000FF;#
    border:1px solid #FF0000;#
    z-index:5;#
}#
```

```
}#
</style>#
```

And to set the pin to display on the map, you would use something like this:

```
function Button1_onclick()#
{#
    map.SetCenter(55, -1.5);#
    map.AddPushpin('pin', 55, -1.5, 88,
        34, 'pin', 'MyPin');#
}#
```

You can remove the pin using the RemovePushpin method. The AddPushpin method returns a DOM object, which is the DIV corresponding to the pin. This allows you to change any of the pushpin's properties should you wish. For example, if you want to create a pushpin in the form of a camera icon to mark where you took a photo, you would define a style similar to this:

```
<style type="text/css" media=screen>#
.pin2#
{#
    width:32px;height:23px;#
    font-family:Arial,sans-serif;#
    font-weight:bold;font-size:8pt;#
    color:White;overflow:hidden;#
    cursor:pointer;text-decoration:
        none;#
    text-align:center;#
    border:0px;#
    background: url(camera.gif);#
    z-index:5;#
}#
</style>#
```

This assumes that there is a file called camera.gif containing a 32x23-pixel camera shape complete with transparent background in the same directory as your web page.

Try this out using something like:

```
map.SetCenter(55, -1.5);#
map.AddPushpin('pin', 55, -1.5, 88, 34,
    'pin2', '');#
```

You should see a pushpin in the shape of a camera. The pushpin moves with the map but doesn't scale if you zoom in or out. That is, the camera icon stays the same size no matter what.

DRAWING ON THE MAP

As the map is just a collection of dynamic HTML elements within a DIV tag generated by a script, we can still use everything we know about DHTML and its extensions to manipulate the map. In particular, we can use Vector Markup Language (VML) to draw on the map.

To use VML, we need to add the VML namespace to the HTML tag:

```
<html xmlns:v="urn:schemas-microsoft-
    com:vml">#
```

Then add a style within the header:

```
<style>#
    v:* { behavior: url(#default#VML);
    }#
</style>#
```



REVIEW BOOK

Excel 2003
Programming: A
Developer's Notebook

RATING ★★★★★

PRICE £21

ISBN 0596007671

PAGES 312



This is a great book, but you won't find it suitable if you want a guide to general Excel programming. Author Jeff Webb's approach is to view Excel 2003 as part of a bigger system comprising XML, SharePoint, InfoPath and web services. It uses VBA for much of the programming, but this isn't a book about programming in VBA. There is a chapter on using .NET language with Excel, but in many ways this is the weakest section. The bulk of the book discusses how XML ties applications together. Sharepoint is used to manage and share data between applications and users, and InfoPath is used to create forms, gather data and save it in XML format. Excel can load the XML data, select items and process it. Using VBA macros, you can also collect XML data by connecting to web services. The book explains everything well and the examples are easy to follow, with useful hints and tips. If you want a book about general VBA programming or Excel programming, this isn't for you. If you want to learn about the exciting world of XML and Excel, though, it is highly recommended.

✓ Well-written and concise guide to Excel and XML programming

✗ Focused on one narrow view of Excel programming

REVIEW BOOK

Programming
.NET Components

RATING ★★★★★

PRICE £32

ISBN 0596007620

PAGES 648



Juval Lowy's book has deservedly reached its second edition. From the title you might expect it to be narrow and to focus on creating components. While it does explain the advantages of the component approach to building applications, it ranges widely over topics that tell you in simple and precise terms how .NET works. It tells you how code is compiled and managed by the system. It explains security, threading, events, interfaces, serialisation, remoting and more. Everything is explained from the point of view of creating components, but it is easily applied to other uses, too.

Most examples are in C# but there are helpful comments about VB .NET, and most of the ideas are applicable to any .NET language. It covers aspects of .NET 1.0, 1.1 and 2.0, and comments on how things have improved and how new facilities should be used. This edition deals with Visual Studio 2005, so even if you own the first edition it's well worth buying this one as well.

This isn't a beginner's book, but it isn't a difficult book either. It is easily the best book on .NET we've encountered so far and much better than the huge brick-sized tomes that state the obvious and rehash the .NET documentation.

✓ In-depth but easy-to-follow description of how .NET works

✗ Not for the complete beginner

Now we can draw vector graphics on the page. For example:

```
<v:oval style='position:absolute;top:50px;
left:10px;width:100pt;
height:75pt;z-index:5'
fillcolor="red">
</v:oval>
```

If you're using Internet Explorer, this will draw a large red ellipse on top of the map. The ellipse will not pan or zoom with the map as it is fixed on the page. However, you can see in principle that it would be possible to move it and scale it with the map, even if the calculations needed to do so are slightly fiddly.

A neater way to add features such as graphics to the map would be to modify MapControl.js, but this is against the terms of the licence. However, it is almost as easy to create a script object that works alongside it. Microsoft provides an example of how this can be done in the form of widgets.

USING WIDGETS

As well as the main MapControl script there is a lesser-known script that provides some user interface controls, or widgets. It can be accessed at <http://virtualearth.msn.com/js/ve.js>. As with MapControl.js you can download this file and use it locally or dynamically link to it within your web pages. To add the script to a web page use:

```
<script src="http://virtualearth.
msn.com/js/ve.js">
</script>
```

One of the most useful widgets is the compass. This uses a bitmap to create a compass control that

moves the map in the direction that the user clicks on. You have to provide the bitmap and if you are going to place it over the map it's a good idea for it to be a GIF with a transparent background. I used a simple arrow symbol measuring 48x48 pixels. As there are a lot of properties to set, it is wise to create the compass widget in a separate function:

```
function CreateCompass()
{
    var el=document.createElement
    ("div");
    el.id="VE_Compass";
    el.style.background="url(compass.
gif)";
    el.onmousedown=VE_Compass._
MouseDown;
    el.onmouseup=VE_Compass._MouseUp;
    el.onmousemove=VE_Compass._
MouseMove;
    el.style.position="relative";
    el.style.top = 10;
    el.style.left = 15;
    el.style.zIndex=31;
    el.style.width = 48;
    el.style.height = 48;
    VE_Compass.element=el;
    document.body.appendChild(el);
}
```

You can see how all this works in a very similar way to the main map control: an object is added to the DOM complete with styles and so on. The width and height specified should be the dimensions of the graphic in pixels. If you specify any other size the graphic is cropped or repeated to fit and no scaling is performed. To add the compass control to the page, you simply call the function after the map control has been created:

```
CreateCompass();
```

As well as a compass control, there is a zoom control, a scale control and a scratchpad. These are used in pretty much the same way as the compass, and details can be found on the Via Virtual Earth website. There is also a search function that will return the location chosen by address or city name, but this only works for US locations.

CONTROLLING DEPENDENCIES

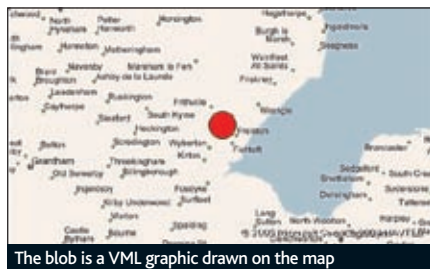
Although you can't legally modify MapControl.js, you can add extra functions by providing scripts that work alongside it. This introduces some dependencies – for instance, we'll assume that the map control will be called 'map' by default – but these are easily accommodated.

You need to delve inside the main script, read it and make use of the knowledge of its inner workings. For example, you can use expressions such as:

```
var y=map._LatToY(53)-map.originY;
var x=map._LonToX(0)-map.originX;
```

These convert latitude and longitude positions into screen positions on the map. This can be used to find the screen coordinates of locations on the map, which you can then draw on, but there's still the big problem of updating these coordinates when the map pans or zooms.

Unless you have a very advanced and clever mapping idea, most of what you'll probably want to do is better achieved using the humble pushpin. The pushpin already provides a mechanism for locating an object on the map and keeping the object in the correct place as the map is zoomed and panned. The key idea in using this is that a pushpin is a DIV object, and almost any other object can be placed



The blob is a VML graphic drawn on the map

inside it. However, let's start with something slightly simpler and work up.

When you create a pin using the `AddPushpin` method, it returns a reference to the DOM object that has just been created. If you keep this reference you can modify the pushpin. In fact, you don't even need to use a CSS style to define a pushpin because you can pass a null string to the method and rely on a default set of style settings.

For example, to place a large, bright red block on the map you can use:

```
var p=map.AddPushpin('pin',53,0, 88,
  34, '', '');
p.style.width=64;
p.style.height=64;
p.style.border=0;
p.style.zIndex=5;
p.style.background="#FF0000";
```

As you might expect, the block scrolls and moves around as if attached to the map.

Now that you have the idea of using pushpins interactively in this way, you can see that they can be thought of as little bits of canvas that you can draw on. Let's see how to add a VML circle object to the pushpin. First we need a pushpin object:

```
var p=map.AddPushpin('pin',53,0, 88,
  34, '', '');
p.style.width=64;
p.style.height=64;
p.style.border=0;
p.style.zIndex=5;
```

Next we create a VML oval object:

```
var el=document.createElement("v:
  oval");
```

This can be customised in the same way:

```
el.id="Circle";
el.style.position="relative";
el.style.top = 0;
el.style.left = 0;
el.style.zIndex=31;
el.style.width = 10;
el.style.height = 10;
el.fillcolor="red";
```

Finally we make the VML oval a child object of the pushpin:

```
p.appendChild(el);
```

This places the VML within the DIV tags that make up the pushpin. Since the position of the circle is defined relative to its parent element, the pushpin, the circle is moved around the map with the pushpin. The only potential problem is that the

circle doesn't scale if the map is zoomed. If you need it to change size it's simplest to put code in the `OnEndZoom` event handler to resize the circle manually.

THE JSCRIPT OBJECT

There is nothing stopping you from packaging the extensions that you have made to the pushpin object as another JScript object. This is also a good example of creating objects in JScript.

To do this, we must define a function that acts as the object's constructor. The code in the constructor is carried out when an instance of the object is created. To create object properties and methods you use the 'this' keyword. A constructor to create a blob-shaped marker might start:

```
function MarkerBlob(lat,lon,h,w,c)
{
  this.lat=lat;
  this.lon=lon;
  this.w=w;
  this.h=h;
  this.c=c;
```

This creates properties `lat`, `lon`, `w`, `h` and `c`, and initialises them to the constructor's parameters. Say an instance of the object is created using:

```
var Blob=new MarkerBlob(53,0,32,32,
  "#FF0000");
```

You can then access the properties using the usual syntax:

```
Blob.lon=20;
```

Of course, changing the value of a property doesn't update the DOM objects associated with the JScript object. In this case, you need to write a `SetLon` function that also updates the DOM object.

THE VML OVAL OBJECT

The next task is to create the VML oval object as before. The only real difference is that the JScript object's property values are used:

```
var el=document.createElement("v:
  oval");
el.id="Circle";
el.style.position="relative";
el.style.top = 0;
el.style.left = 0;
el.style.zIndex=31;
el.style.width = this.w;
el.style.height = this.h;
el.fillcolor=this.c;
this.el=el;
```

The final line creates an 'el' property that stores the entire DOM object. This has to be updated to affect the DOM object displayed.

The final step is to create the pushpin and store its details within the JScript object:

```
this.pin=map.AddPushpin('pin',this.
  lat,
  this.lon, this.w, this.h, '', '');
this.pin.style.border=0;
this.pin.style.zIndex=5;
this.pin.appendChild(el);
}
```

Once again, the details of the DOM object corresponding to the pin are stored as a property of the `MarkerBlob` object.

Now that the `MarkerBlob` class is complete, we can create an instance using:

```
var Blob=new MarkerBlob(53,0,32,32,"#
  FF0000");
```

This is just the start. Now we have an object constructor, we can start to add methods and additional properties that add features. For example, to set the latitude and longitude to a new value, we define a new method:

```
this.SetLatLon=function(lat,lon)
{
  this.lat=lat;
  this.lon=lon;
  for(var i=0;i<map.pushpins.length;
    i++)
  {
    var p=map.pushpins[i];
    if (p.id=="pin")
    {
      p.lon=lon;
      p.lat=lat;
    }
  }
  map._RepositionPushpins();
}
```

Once this is defined we can call the method using:

```
Blob.SetLatLon(55,-1.5);
```

The details of the method are slightly more complicated than you might expect, because while we have a reference to the pushpin as a DOM object, we don't have a reference to it as an object within the map. Fortunately, the details of all the pins in use are stored in an array. This allows us to scan the array for our pin using its ID. Once the pin is found, we change its latitude and longitude then use the internal function `_RepositionPushpins` to reposition all the DOM objects that correspond to the JScript pins.

THE POINT OF IT ALL

You can use similar techniques to create all manner of map markers and overlays. You can even use pushpins to draw lines that move with the map. To do this, you need to remember that only the top left-hand corner of the pushpin is fixed at a given latitude and longitude – the other corner moves according to the zoom in use. To get two geographically fixed points at the start and end of a line you simply use two pushpins and connect their top left-hand corners. The lines must be updated each time the map is panned or zoomed.

A careful study of the map control yields lots of good ideas and techniques – it's almost a free textbook. [ES](http://www.computershopper.co.uk)

CONTACT

MIKE JAMES

Mike has written several books on computing and programming and has been a senior lecturer at Teesside University

mike@computershopper.co.uk